



Mobile devices as development platform in Broken Age

Oliver Franzke

Lead Programmer, Double Fine Productions



p1xelcoder

GAME DEVELOPERS CONFERENCE®

MOSCONE CENTER · SAN FRANCISCO, CA

MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015



What makes a dev platform?

- Fast iteration time

- Content



- Native code



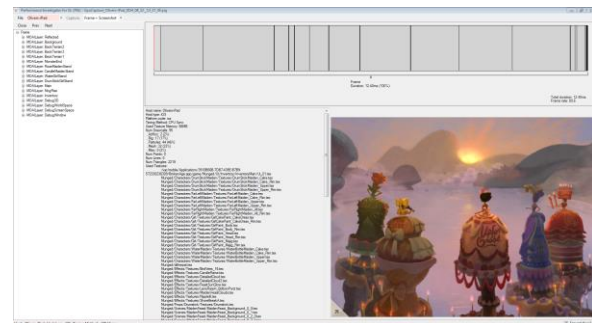
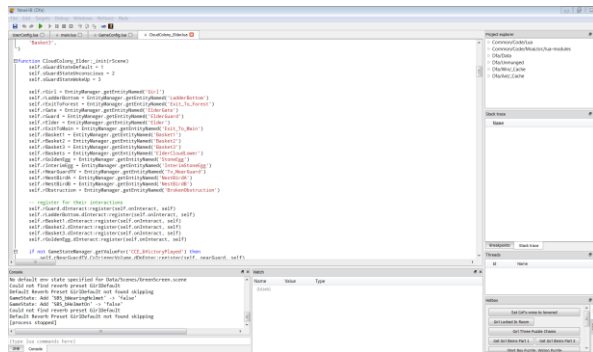
- Debuggability

- Game-play script



- Native code

- Graphics



BROKEN AGE



PS4

PSVITA

PlayStation Vita






Developing BA on mobile devices

- Some of the stuff we did on device
 - Game-play programmer wrote touch controls
 - VFX artist checked and optimized effects
 - Fix shader bugs

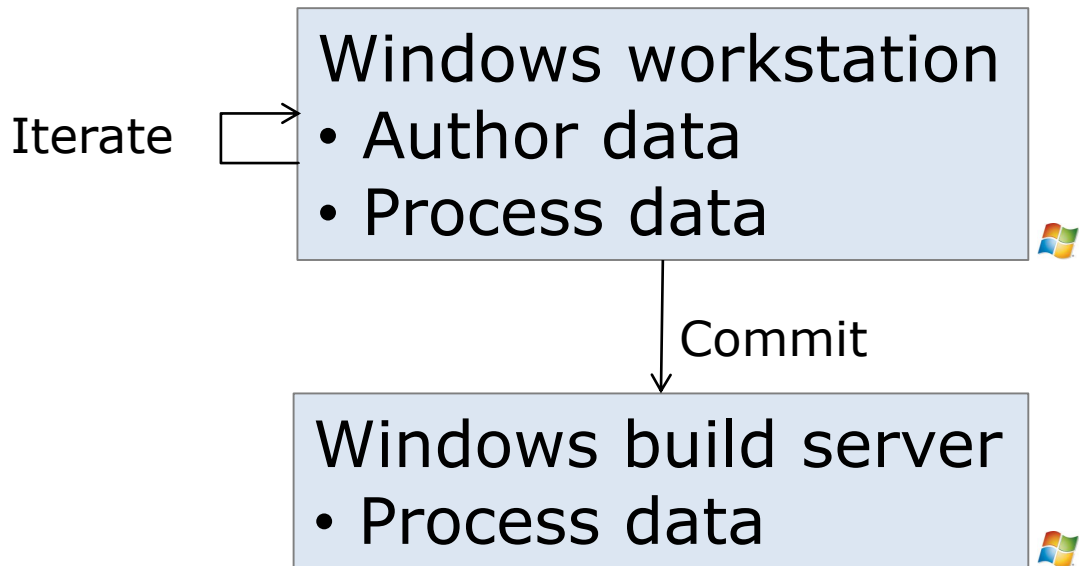


Double Fine development pipeline

- Target platform:   
 - Code: compiled on target platform
 - Data: shared, authored on Windows
- Same hardware
 - Performance characteristics similar
 - No per-platform content





Content workflow



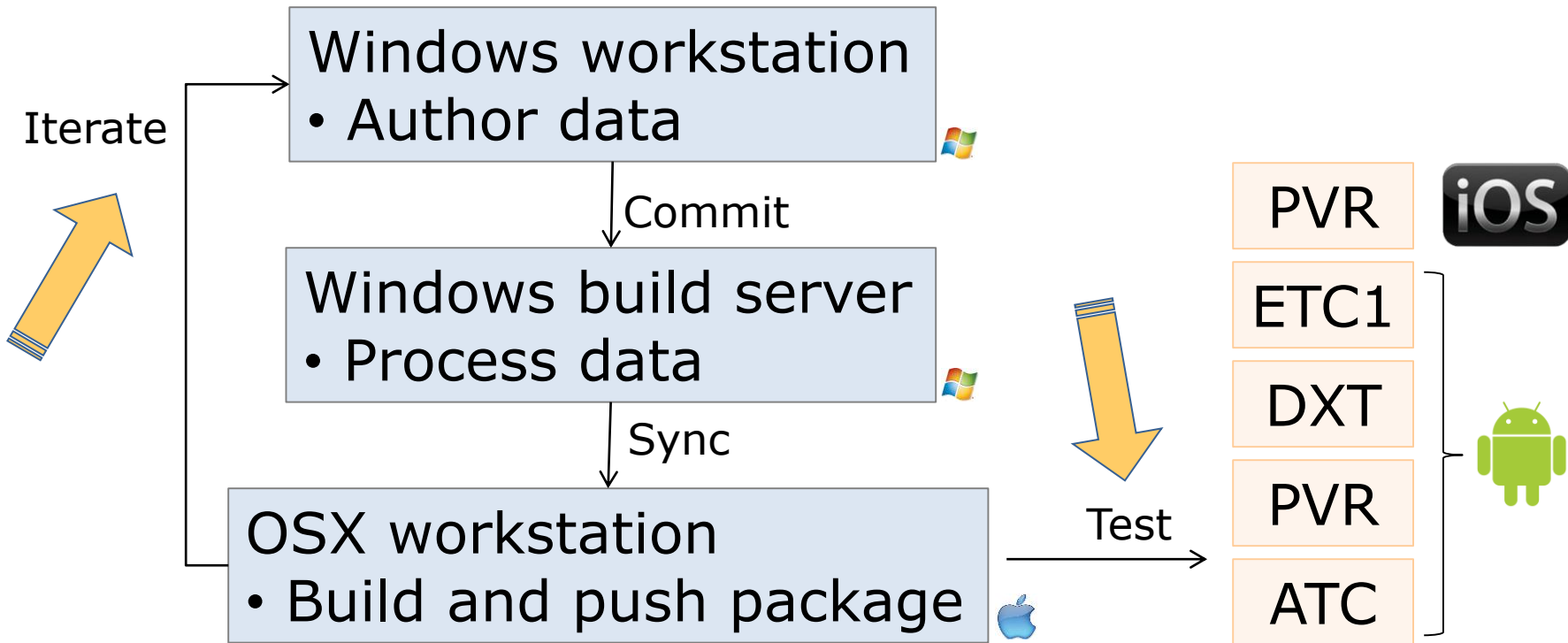


Double Fine development pipeline

- Target platform:  
 - Code: compiled on OSX
 - Data: **not shared**, authored on Windows
- Hardware is very different
 - Performance varies greatly
 - Platform / GPU specific data necessary





Content workflow





Mobile as development platform


- Broken Age Act 1 package size: ~1.2GB
- Reboot time was ridiculous!
 -  ~10 minutes*
 -  8 – 21 minutes
- Infeasible for development!

* Recent Xcode update reduced reboot time to 2 – 3 minutes






Too slow

- What is going on? 
 - Relink and update staging data: 10+ seconds
 - Data sync and install: ~10 minutes
- Sync (comparison and copy) is bottleneck





Incomprehensibly slow

- What is going on? 
 - Package build time: 4+ minutes
 - Data transfer and install: 4 - 17 minutes
- USB speed is bottleneck





Slow content update

- Same symptom but different cause
- Different solutions required
 -  Minimize APK size
 -  Work around Xcode data sync



Fast content update



- Minimal APK: No data, just code
 - Reduced package build time
 - Fast APK transfer and installation
- Deal with data separately
 - Only copy added or changed files



Fast content update

- Load assets from 'sdcard'
 - Supported by (almost) all Android devices
 - Remap file location

```
bool RemapFilename( const char* filename, char* remapped ) {  
    #if _DEV  
        sprintf(remapped, "/sdcard/dfp/dfa/%s", filename);  
        return FileExists(remapped);  
    #else  
        return false;  
    #endif  
}
```



Fast content update



- Data sync
 - 1st approach: Consistent file database
 - Sync changes using ADB (e.g. adb push ...)
 - Keep track of files on device
 - Update database during sync
 - Slow and inconvenient
 - Multiple devices: Per-device database?!



Fast content update



- Data sync

- 2nd approach: Scan device files

- Naïve implementation is sllloooooowwww....

- Re-implement ADB protocol based on OS source

- https://android.googlesource.com/platform/system/core.git/+/_master/adb/





Fast content update



- Data sync
 - 2nd approach: Scan device files (cont.)
 - Example: List directory

```
socket.send(pack("LIST", 15, "/sdcard/dfp/dfa"))
while True:
    id, mode, size, time, namelen = unpack(socket.recv(16))
    name = '' if namelen == 0 else _recvall(socket, namelen)
    if id == "DONE": break
    if stat.S_ISDIR(mode):
        dirs.append(...)
    elif stat.S_ISREG(mode):
        files.append(...)
return (dirs, files)
```



Fast content update



- Data sync
 - 2nd approach: Scan device files (cont.)
 - Compare files and compute diff
 - Sync files using ADB protocol
 - Very fast
 - No database



Fast content update



- Results
 - Data sync: 15 - 25 seconds
 - APK build, copy and install: 30 - 40 seconds
- 10 – 20+ speedup!





Fast content update

- Minimal staging folder
 - Delete unchanged files
 - Copy only added or changed files
 - Use sync timestamp
- Reduced work for Xcode
- No run-time changes necessary



Fast content update

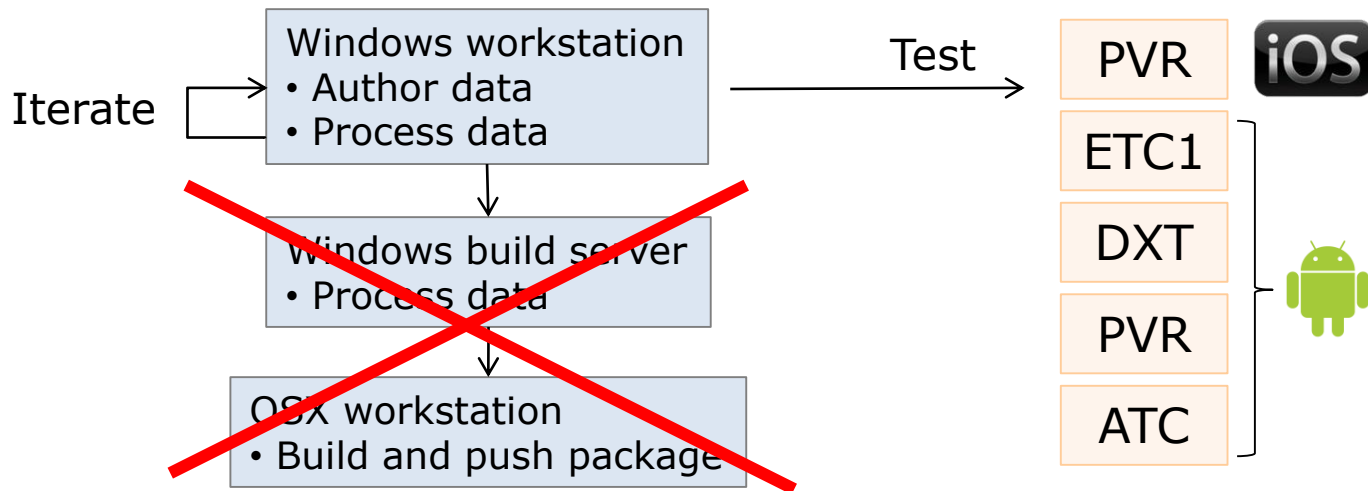
- Results
 - Relink and update staging data: 15+ seconds
 - Data sync and install: 30 seconds
- 13 speedup (latest Xcode 2 – 4 speedup)





Mobile as development platform

- Ideal workflow





File streaming

- No re-sync necessary*
- Hot-reload changed files
- No OSX workstation needed
- Artists can work on target device

* Code changes require re-sync



File streaming

Client

Game session on device



`fopen()`, `fread()` ...

Manager

Worker thread 1

Worker thread 2

Server

Python-based tool on workstation



`send()`, `recv()`

Server thread 1

Server thread 2

Script file A

DXT file B

DXT file C

PVR file D



File streaming

- Latency optimization
 - Use local file cache

Client

Game session on device



`fread(out, 64, 1, fp)`

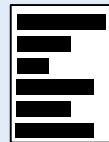
Cache miss ☹️



File cache

Server

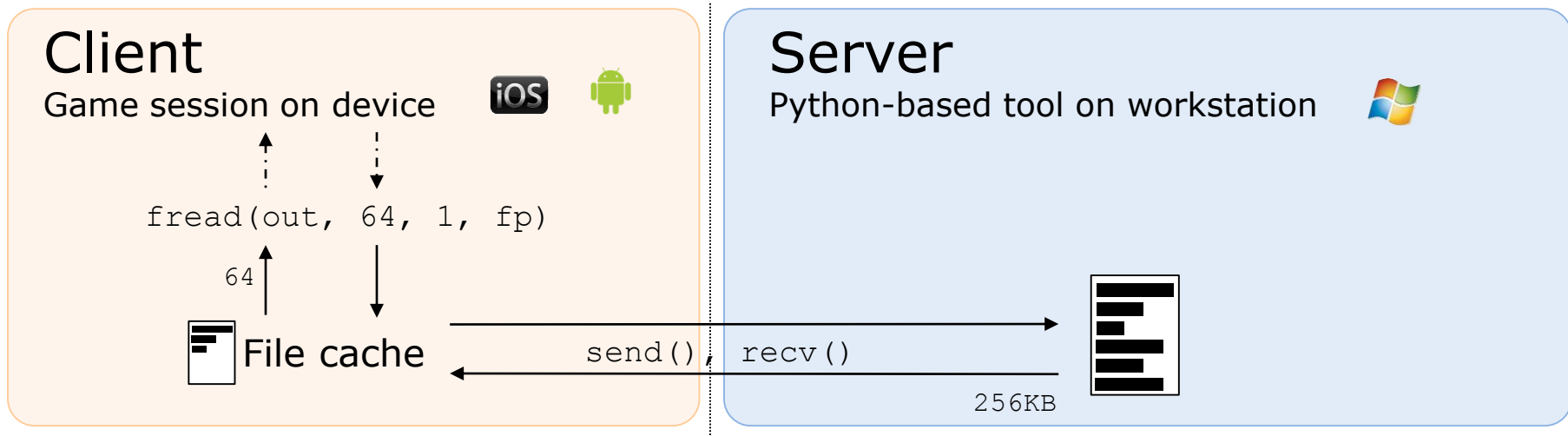
Python-based tool on workstation





File streaming

- Latency optimization (cont.)
 - Read at least one cache line





File streaming

- Latency optimization (cont.)
 - Exploit data locality

Client

Game session on device



`fread(out, 4, 2, fp)`

Cache hit 😊

8



File cache

?

Server

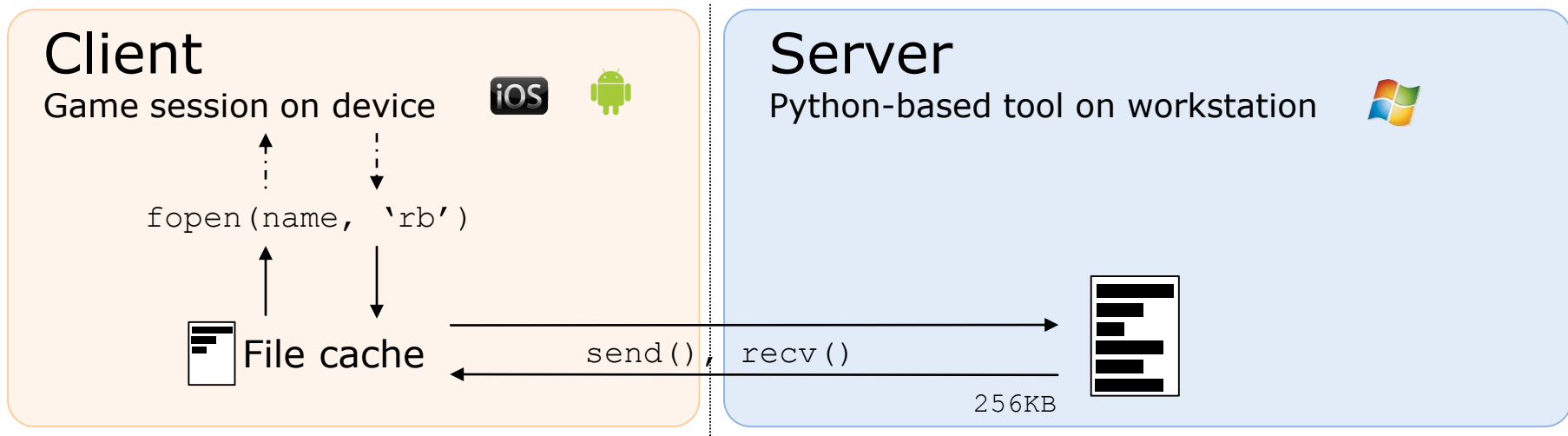
Python-based tool on workstation





File streaming

- Latency optimization (cont.)
 - `fstat()` & `fopen()` read first cache line





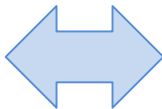
File streaming

- Latency optimization (cont.)
 - Cache expires after 5 ms to avoid stale data
 - Multiple concurrent requests
 - User filter to define streamed files
 - Local IO will always be faster





- Fully integrated into our game editor 2HB

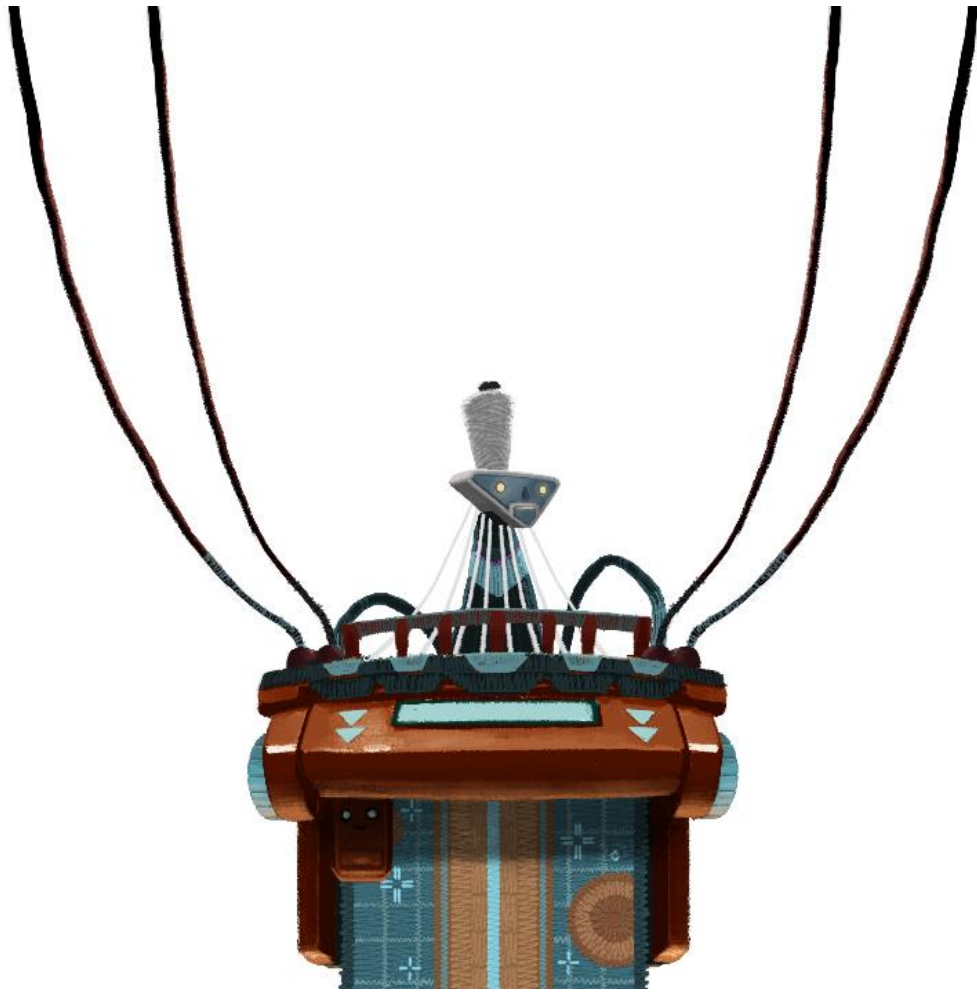




Conclusion

- Mobile as development platform is...
 - ...not trivial
 - ...worth your time
 - ...necessary for ~~a big~~ project
every





Thank you!

Questions?



p1xelcoder